باسمه تعالی
دانشگاه تهران - دانشکده مهندسی برق و کامپیوتر

ساختمان‌های داده و الگوریتم

تمرین چهارم - مرتب‌سازی و درهم‌سازی

محمد امانلو، کوروش سجادی

تاریخ تحویل: ۱۴۰۳/۰۲/۳۰

۱. مرتب سازی جفتی — ۱۰ نمره

فرض کنید A آرایه‌ای از n جفت عدد صحیح مثبت (x, y) که $x_i, y_i < n^۲$ برای هر $i \in \{۰, ..., n-۱\}$ باشد. توان جفت (x, y) عدد صحیح $x + n^y$ است. یک الگوریتم با مرتبه زمانی $O(n)$ برای مرتب‌سازی جفت‌ها در A به صورت صعودی و بر اساس توان آنها توصیف کنید.

پاسخ :

ابتدا توجه داشته باشید که برای هر عدد صحیح $y > ۱$ و برای هر $x \in \{۰, ..., n^۲-۱\}$ رابطه $x < n^y$ صحیح است. آرایه A را پیمایش کنید و تمام جفت‌های دارای $y = ۱$ را در آرایه $A_۱$ و تمام جفت‌های دیگر را در آرایه $A_۲$ قرار دهید. $A_۱$ را مستقیماً با محاسبه و مقایسه توان مربوطه $x + n$ مرتب کنید. از آنجایی که این مقادیر حد بالای $O(n^۲)$ دارند، $A_۱$ را در زمان $O(n)$ با استفاده از مرتب سازی $Radix$ مرتب کنید. برای مرتب‌سازی $A_۲$ از مرتب‌سازی $tuple$ استفاده کنید، ابتدا بر اساس مقادیر x و سپس بر اساس مقادیر y مرتب‌سازی کنید (زیرا توان به تغییرات y حساس‌تر از است).از آنجایی که مقادیر x و y هر دو از بالا با $O(n^۲)$ محدود شده‌اند، می‌توانیم از مرتب‌سازی $Radix$ برای مرتب‌سازی $A_۲$ به صورت پایدار در زمان $O(n)$ استفاده کنیم. سپس $A_۱$ و $A_۲$ را در زمان $O(n)$ با استفاده از مرحله $merge$ در مرتب سازی $merge$ در زمان $O(n)$ ادغام کنید.

۲. سلام بر فیثاغورث — ۱۰ نمره

یک $Quad$ فیثاغورثی از چهار عدد صحیح (a, b, c, d) تشکیل شده است به طوری که $d = \sqrt{a^۲ + b^۲ + c^۲}$ باشد. آرایه A حاوی n عدد صحیح مثبت متمایز است. الگوریتمی با مرتبه زمانی متوسط $O(n^۲)$ به گونه ای توصیف کنید تا تعیین کند آیا چهار عدد صحیح از A یک $Quad$ فیثاغورثی را تشکیل می دهند یا خیر. اعداد صحیح از A ممکن است بیش از یک بار در $Quad$ ظاهر شوند.

پاسخ :

ابتدا مشاهده می کنیم که کافی است (a, b, c, d) را پیدا کنیم تا $a^۲ + b^۲ = d^۲ - c^۲$. فرض کنید P مجموعه ای از $n^۲$ جفت اعداد صحیح مرتب شده از A باشد، که در آن اعداد صحیح در A ممکن است در یک جفت تکرار شوند. یک جدول هش خالی H بسازید و برای هر جفت $(a, b) \in P$، مقدار $a^۲ + b^۲$ را محاسبه کرده و در H وارد کنید. سپس برای هر جفت $(c, d) \in P$.مقدار $d^۲ - c^۲$ را در H محاسبه و جستجو کنید اگر مقدار در H باشد، مقدار $a^۲ + b^۲$ برابر است با مقدار $d^۲ - c^۲$، بنابراین برگردانید که $Quad$ فیثاغورثی وجود دارد. در غیر این صورت. اگر $d^۲ - c^۲$ در H وجود نداشته باشد، برگردانید که $Quad$ فیثاغورثی وجود ندارد. هر مقدار $a^۲ + b^۲$ یا $d^۲ - c^۲$ برای محاسبه زمان ثابتی نیاز دارد. بنابراین محاسبه همه آنها در بدترین حالت به زمان $O(n^۲)$ نیاز دارد، در حالی که قرار دادن آنها در جدول هش یا جستجوی آنها در جدول هش زمان مورد انتظار $O(n^۲)$ را می طلبد. بنابراین این الگوریتم در مجموع در زمان $O(n^۲)$ به طور متوسط اجرا می شود.

۳. لقمه حیا ۱۳ نمره

در یک مهمانی، آقای چاق قصد دارد بزرگترین میوه را انتخاب کند. اما چون می‌داند برداشتن بزرگترین میوه به مذاق صاحبخانه خوش نمی‌آید قصد دارم دومین بزرگترین میوه را انتخاب کند. ثابت کنید او در بدترین حالت با انجام $n + \lceil lg(n) \rceil - ۲$ مقایسه می‌تواند دومین بزرگترین میوه را پیدا کند.

پاسخ :

برای پیدا کردن دومین بزرگترین میوه، می‌توان به صورت یک مسابقه بازی کرد. ابتدا تمامی میوه‌ها را در جفت‌ها با هم مقایسه می‌کنیم تا بزرگترین میوه‌ها به جمعیت بزرگترین‌ها بپیوندند، سپس از جمعیت بزرگترین‌ها دومین بزرگترین میوه را پیدا کرده و از آن مقایسات هم برای بدست آوردن دومین بزرگترین میوه استفاده می‌کنیم.

مقایسات میان میوه‌ها به این صورت پیش می‌رود که تمامی میوه‌ها به تعداد ۲ تا مقایسه می‌شوند تا بزرگترین میوه هر گروه را پیدا کنیم. سپس بزرگترین میوه‌ها به تعداد $\lceil \frac{n}{۲} \rceil$ مقایسه می‌شوند تا دومین بزرگترین میوه را پیدا کنیم.

از آنجا که هر میوه به صورت دقیقا یک بار بازنده در یک مقایسه است، باید از هر میوه به تعداد یک بار مقایسه شود تا دومین بزرگترین میوه را پیدا کنیم.

در نهایت، تعداد کل مقایسات برای پیدا کردن دومین بزرگترین میوه برابر است با:

$$n - ۱ + \lceil \log n \rceil - ۱ = n + \lceil \log n \rceil - ۲$$

بنابراین، آقای چاق در بدترین حالت با انجام $n + \lceil \log n \rceil - ۲$ مقایسه می‌تواند دومین بزرگترین میوه را پیدا کند.

۴. دژ مرموز ۱۵ نمره

فرض کنید قهرمانی به نام کسرا در یک دژ مرموز به دام افتاده است. در هر اتاق این دژ، یک صندوق وجود دارد که فقط زمانی باز می‌شود که کسرا توپ‌های جادویی را در ترتیب صعودی قرار دهد. این توپ‌ها در ابتدا به صورت نامرتب در صندوق قرار دارند. کسرا تنها می‌تواند توپ‌های مجاور را جابجا کند، به شرطی که تفاوت بین دو توپ فقط یک واحد باشد. الگوریتمی از مرتبه زمانی $O(n)$ ارائه دهید که به کسرا بگوید این کار قابل انجام است یا خیر.

مثال:

ورودی: $arr[] = \{۲, ۱, ۵, ۴\}$
خروجی: بله
توضیح: با جابجایی ۲ و ۱ در یک مرحله و ۴ و ۵ در مرحله دیگر آرایه سورت می‌شود.

پاسخ :

هدف این است که آرایه را با جابجا کردن فقط عناصر مجاور، به ترتیب صعودی مرتب کنیم. برای اینکه توپ‌ها را مرتب کنیم، کسرا باید تفاوت عناصر مجاور را چک کند و اطمینان حاصل کند که تفاوت آنها فقط یک واحد است.

۱. ما بررسی می‌کنیم که آیا کل آرایه قبل از هر اندیس i به درستی مرتب شده است. این کار را با محاسبه بیشینه $\text{arr}[۰..i - ۱]$ انجام می‌دهیم و اطمینان حاصل می‌کنیم که این بیشینه یا کوچکتر از $\text{arr}[i]$ باشد یا فقط یک واحد بیشتر از $\text{arr}[i]$ باشد. در این حالت اول، فقط به سمت جلو حرکت می‌کنیم.

۲. اگر عنصر کنونی $\text{arr}[i]$ از عنصر بعدی $\text{arr}[i + ۱]$ بزرگ‌تر است، موارد زیر را بررسی می‌کنیم:

- اگر تفاوت بین $\text{arr}[i]$ و $\text{arr}[i + ۱]$ فقط یک واحد است، آن دو را مبادله می‌کنیم.
- اگر تفاوت بیشتر از یک واحد باشد، این نشان می‌دهد که مرتب‌سازی آرایه با جابجایی‌های مجاور امکان‌پذیر نیست و در نتیجه ‘false‘ برمی‌گردانیم.

۳. اگر تفاوت بین دو عنصر مجاور $\text{arr}[i]$ و $\text{arr}[i+1]$ بیشتر از یک واحد باشد، این بدان معناست که ترتیب صحیح آنها نمی‌تواند فقط با یک مبادله ساده برقرار شود. برای مثال، اگر دو عنصر مجاور ۲ و ۴ باشند، ما نمی‌توانیم آنها را فقط با یک مبادله به ترتیب صعودی درست تبدیل کنیم، چرا که هر جابجایی مجاور تنها می‌تواند ترتیب دو عنصر مجاور را تغییر دهد و نمی‌تواند تفاوت بزرگ‌تر از یک واحد را پوشش دهد. در نتیجه، اگر چنین وضعیتی پیش بیاید، به این معناست که مرتب‌سازی آرایه با محدودیت مبادله فقط عناصر مجاور امکان‌پذیر نیست و بنابراین 'false' را برمی‌گردانیم.

۴. اگر تا انتهای آرایه بدون برخورد به مشکل پیش رفتیم و تمام مقایسه‌ها و جابجایی‌ها با موفقیت انجام شده باشند، نتیجه‌گیری می‌کنیم که مرتب‌سازی آرایه امکان‌پذیر است و 'true' را برمی‌گردانیم.

۵. بازی ترند — ۱۷ نمره

در یک بازی جدید که اخیراً محبوب شده است، بازیکنان باید به سرعت یک سلسله از اعداد طبیعی را بازیابی کنند که به هم ربط دارند ولی ممکن است در ترتیب پراکنده‌ای در میان دیگر اعداد قرار گرفته باشند. شما به عنوان یک برنامه‌نویس می‌خواهید ابزاری بسازید که طولانی‌ترین دنباله متوالی از اعداد را در یک لیست معین شناسایی کند.

Input: $arr[] = ۱۵, ۱۲, ۱۳, ۹, ۱۱, ۱۴$

Output: ۵

توضیحات: اعداد متوالی در این زیردنباله شامل {۱۲، ۱۵ ، ۱۳، ۱۴، ۱۱} هستند که می‌توانند پشت سر هم قرار گیرند و طول آن ۵ است.

پاسخ :

برای حل این مسئله، مراحل زیر را دنبال کنید:

۱. یک هش خالی ایجاد کنید.

۲. تمام عناصر آرایه را در هش وارد کنید.

۳. برای هر عنصر `arr[i]` در آرایه، موارد زیر را انجام دهید:

(آ) بررسی کنید که آیا این عنصر نقطه شروع یک زیردنباله است. برای بررسی این موضوع، کافی است به دنبال `arr[i] - 1` در هش بگردید. اگر پیدا نشد، پس این اولین عنصر یک زیردنباله است.

(ب) اگر این عنصر، اولین عنصر باشد، سپس تعداد عناصر متوالی را که با این عنصر شروع می‌شوند، بشمارید. از `arr[i] + 1` شروع کنید تا آخرین عنصری که می‌توانید پیدا کنید.

(ج) اگر تعداد بیشتر از بلندترین زیردنباله قبلی پیدا شده باشد، سپس آن را به‌روزرسانی کنید.

عملکرد الگوریتم مبتنی بر این فرض است که برای تشخیص ابتدای یک زیردنباله پیوسته جدید، ما باید وجود عدد پیش از عدد فعلی (مثلا `arr[i] - 1`) در هش را بررسی کنیم. این کار به ما کمک می‌کند تا تشخیص دهیم آیا عدد فعلی می‌تواند ابتدای یک دنباله جدید باشد یا خیر. اگر عدد `arr[i] - 1` در هش نباشد، این به این معناست که `arr[i]` می‌تواند ابتدای یک دنباله جدید باشد و ما شروع به شمردن طول این دنباله می‌کنیم. با پیشروی در اعداد `arr[i] + 1`، `arr[i] + 2`، و غیره، می‌توانیم طول دنباله را افزایش دهیم تا جایی که دیگر نتوانیم عدد بعدی را در هش پیدا کنیم. این فرایند به ما امکان می‌دهد که بلندترین زیردنباله پیوسته را شناسایی کنیم. اگر طول این زیردنباله بیشتر از بلندترین زیردنباله‌ای باشد که تاکنون شناسایی شده، آن را به‌روزرسانی می‌کنیم. در نهایت، این روش به ما کمک می‌کند تا با کارایی بالا و بدون نیاز به مرتب‌سازی اولیه، بلندترین زیردنباله پیوسته را در یک آرایه پیدا کنیم.

۶. قبیله بازیگوش — ۱۸ نمره

فرض کنید در یک جزیره دورافتاده، قبیله‌ای وجود دارد که بازی سنتی خود را دارند. این بازی به این صورت است که شرکت‌کنندگان کارت‌هایی را در یک ردیف قرار می‌دهند و سپس سعی می‌کنند تعداد حرکات لازم برای تبدیل ردیف کارت‌ها به یک ردیف کاملاً مرتب‌شده

را حساب کنند. هر حرکت شامل جابجایی دو کارت است. هدف این است که با کمترین تعداد جابجایی، کارت‌ها را مرتب کنند. الگوریتمی ارائه دهید که با مرتبه زمانی $O(n \log n)$ بتواند تعداد این جابجایی‌ها را محاسبه کند. اگر کارت‌ها کاملاً مرتب باشند، خروجی باید ۰ باشد و اگر در جهت عکس مرتب باشند، خروجی باید ماکسیمم باشد.

برای فهم بهتر، بیایید یک مثال عینی از یک آرایه کاملا برعکس را بررسی کنیم. فرض کنید آرایه ما به صورت زیر باشد: $[۵, ۴, ۳, ۲, ۱]$. در این حالت، آرایه کاملا برعکس مرتب شده است و تعداد وارونگی‌ها بیشترین مقدار ممکن خواهد بود.

تعداد وارونگی‌ها در این آرایه به صورت زیر محاسبه می‌شود:

- بین ۵ و هر عدد دیگری که پس از آن می‌آید (۴، ۳، ۲، ۱)، وارونگی وجود دارد. (۴ وارونگی)
- بین ۴ و هر عدد دیگری که پس از آن می‌آید (۳، ۲، ۱)، وارونگی وجود دارد. (۳ وارونگی)
- بین ۳ و هر عدد دیگری که پس از آن می‌آید (۲، ۱)، وارونگی وجود دارد. (۲ وارونگی)
- بین ۲ و عدد ۱ که پس از آن می‌آید، وارونگی وجود دارد. (۱ وارونگی)

بنابراین، کل تعداد وارونگی‌ها برای این آرایه برابر با $۴ + ۳ + ۲ + ۱ = ۱۰$ است.

پاسخ :

می‌توان با تغییر دادن Merge-Sort این کار را انجام داد. آرایه را تا رسیدن به حالت پایه می‌شکنیم و تابع Merge خود را این‌گونه تغییر می‌دهیم: اگر $a[i]$ بزرگتر از $a[j]$ باشد، $(\text{mid}i)$ وارونگی وجود دارد زیرا زیرآرایه‌های چپ و راست مرتب شده‌اند، بنابراین تمام عناصر باقی‌مانده در زیرآرایه چپ بزرگتر از $a[j]$ خواهد بود. در هر هنگام تعداد وارونگی‌های چپ، راست و وسط حساب شده را جمع می‌کنیم و در آخر برمی‌گردانیم.

در روند merge، فرض کنید i برای اندیس‌بندی زیرآرایه چپ و j برای زیرآرایه راست استفاده می‌شود. در هر مرحله از merge، اگر $a[i]$ بزرگتر از $a[j]$ باشد، پس $(\text{mid}i)$ وارونگی وجود دارد. زیرا زیرآرایه‌های چپ و راست مرتب شده‌اند، پس تمام عناصر باقی‌مانده در زیرآرایه چپ ($a[i + ۱], a[i + ۲], \ldots , a[\text{mid}]$) بزرگتر از $a[j]$ خواهند بود.

برای پیاده‌سازی این ایده، مراحل زیر را دنبال کنید:

۱. آرایه را به دو نیمه مساوی یا تقریبا مساوی در هر مرحله تقسیم کنید تا به حالت پایه برسید.

۲. تابع merge ایجاد کنید که تعداد وارونگی‌ها را هنگام ادغام دو نیمه از آرایه محاسبه کند.

۳. دو شاخص i و j ایجاد کنید، i برای نیمه اول و j برای نیمه دوم.

۴. تابع بازگشتی ایجاد کنید که آرایه را به نیم‌ها تقسیم کرده و با جمع‌بندی تعداد وارونگی‌ها در نیمه اول، تعداد وارونگی‌ها در نیمه دوم و تعداد وارونگی‌ها با ادغام دو نیمه پاسخ را پیدا کند.

۵. حالت پایه برای بازگشت زمانی است که تنها یک عنصر در نیمه داده شده وجود دارد.

۷. راز اعداد قدیمی　　۲۲ نمره

تصور کنید در یک دهکده کوچک، صندوقچه‌ای از اعداد قدیمی کشف شده است که شامل یک سری از اعداد صحیح به طول n است. مأموریت شما این است که تنها با یک بار دیدن این اعداد تعادل خیر و شر را بازگردانید. الگوریتمی با پیچیدگی $O(n)$ ارائه دهید که طول بزرگترین زیرآرایه از این آرایه را پیدا کند که جمع اعداد آن زیرآرایه برابر با صفر باشد. این زیرآرایه نماینگر تعادل میان خیر و شر در دهکده‌های باستانی است.

پاسخ :

برای حل این ماجراجویی، ما از یک روش جادویی استفاده می‌کنیم که به وسیله سه متغیر اصلی پشتیبانی می‌شود:

- sumCurrent - نشان‌دهنده جمع عناصر آرایه از ابتدا تا جایی که هستیم.
- lengthMax - نشان‌دهنده طول بزرگترین بازه‌ای که جمع آن صفر می‌شود.
- tableHash - یک جدول هش که کلید آن sumCurrent و مقدار آن اندکس i است.

روند کار به این صورت است که بر روی آرایه پیمایش می‌کنیم. اگر به جایی رسیدیم که $\text{sumCurrent} = ۰$ شد، بررسی می‌کنیم که آیا $i + ۱$ بزرگ‌تر از lengthMax است. اگر بله، پس $\text{lengthMax} = i + ۱$ را به‌روزرسانی می‌کنیم. در غیر این صورت، sumCurrent را در جدول هش جستجو می‌کنیم. اگر در جدول وجود داشته باشد و مقدار آن برابر با j باشد، این به معنای آن است که جمع عناصر از j تا i برابر با صفر می‌باشد. پس اگر این بازه از lengthMax بزرگ‌تر بود، آن را به‌روزرسانی می‌کنیم. اگر sumCurrent در جدول هش نبود، آن را در جدول هش درج می‌کنیم. در آخر، lengthMax برابر جواب مسئله است.

فرض کنیم که جمع پیش‌فرض آرایه تا اندیس i را به عنوان S_i نمایش دهیم. حال دو اندیس i و j (که $j > i$) را در نظر بگیرید به گونه‌ای که $S_i = S_j$ باشد.

$$S_i = \text{arr}[۰] + \text{arr}[۱] + \cdots + \text{arr}[i]$$

$$S_j = \text{arr}[۰] + \text{arr}[۱] + \cdots + \text{arr}[i] + \text{arr}[i + ۱] + \cdots + \text{arr}[j]$$

اکنون اگر S_i را از S_j کم کنیم:

$$S_j - S_i = (\text{arr}[۰] + \text{arr}[۱] + \cdots + \text{arr}[i] + \text{arr}[i + ۱] + \cdots + \text{arr}[j]) - (\text{arr}[۰] + \text{arr}[۱] + \cdots + \text{arr}[i])$$

$$۰ = (\text{arr}[۰] - \text{arr}[۰]) + (\text{arr}[۱] - \text{arr}[۱]) + \cdots + (\text{arr}[i] - \text{arr}[i]) + \text{arr}[i + ۱] + \text{arr}[i + ۲] + \cdots + \text{arr}[j]$$

$$۰ = \text{arr}[i + ۱] + \text{arr}[i + ۲] + \cdots + \text{arr}[j]$$

پس می‌بینیم که اگر دو اندیس i و j (که $j > i$) وجود داشته باشند که جمع پیش‌فرض آن‌ها یکسان باشد، زیرآرایه از $i + ۱$ تا j جمع برابر با صفر دارد.

ما می‌توانیم از یک جدول هش برای ذخیره‌سازی جمع پیش‌فرض استفاده کنیم، و اگر به اندیسی رسیدیم که قبلاً یک جمع پیش‌فرض با همان مقدار وجود داشته باشد، یک زیرآرایه با جمع صفر پیدا می‌کنیم. طول این زیرآرایه را با طول بزرگترین زیرآرایه فعلی مقایسه کرده و مقدار بیشینه را به‌روزرسانی می‌کنیم.